1. **Brief Introduction**

This desktop share software is divided into two parts: the server side Control Center and the client Peer. In fact, each peer contains both RDP server as sharer and RDP client as viewer, meaning it has all needed to sharer desktop and view other’s desktop. The Control Center only plays the role of controlling peers with socket commands.

1. **Control Center**
   1. Architecture in general

“Control Center” is the name of the server side. Control Center is organized in three layers in logic. The bottom layer contains two classes derived from CSocket: CDoorSocket and CListenSock, which communicate with client with socket. The mid layer is just one service class CService, providing basic socket services such as sending data and receiving data. This class servers the UI classes from upper layer, sending them messages when some socket commands or state strings come. It also has a list and a map to contain the sockets corresponding to the client. The top layer contains a lot of UI classes and some related helper classes. The UI classes are mainly dialog classes derived from CDialogEx.

* 1. Detail of CService

As mentioned above, the CService class mainly servers UI classes by providing socket services.

Main methods:

SockSend: used to send commands to the client wherever needed.

OnSockReceive: This method will be called once the sockets receive commands or reply to commands from clients. And it will take actions to change the state of the sockets in the map and/or send self-defined messages to corresponding classes according to different received strings. This method is like a message dispatcher.

This class also has a list and a map as its attributes. They are both used to store sockets connected to the clients. But why two containers? The situation is when a client is connected to the Control Center, a new instance of CDoorSocket will be created in method OnSockAccept, later when the socket send a “logname” command along with a name identifying the client itself, the name-socket pair will be added to the map. And in later period only the entity in the map will be used and usually by the name as an id.

As far as the function of this class, the instance of it should be only, which means a singleton. Thus letting it be an attribute of the main dialog is far from enough, I make it an attribute of CControlCenterApp, so that any class can use it though the single “theApp”.

Note that this class also has some pointers to CWnd. They are used to indicate where message be sent in method OnSockReceive. Apparently this is not a good design. And in order to pass some parameters while sending messages, I used the SendMessage function. It might have bad influence on performance.

* 1. Design of UI

The software is a MFC Dialog project. Thus the user interfaces are sets of dialogs and controls on them. The main window is a dialog whose class is CControlCenterDlg , which contains such elements: three dialogs in the client area of the tab control, the static control in the left bottom corner and a corresponding hid “Hand Message” dialog and finally the static control representing number of online clients in the right bottom corner.

* 1. About the CPeerTreeDlg

This class might be the most complex one even though it is just a dialog containing a tree control. I want it to be a portable control so that I set the border of the dialog resource as “none”.

The most basic function of it is to show a tree containing peers divided into different groups. Note that there are two ways to initially construct the tree. The first way is through “BuildTree ” method, firstly read the contents of a set of files from a directory into memory, in the form of a map<CString, list<CString>\* >, whose first element is group name, while the second is the pointer to the list of peer names. The default directory is “.\\Group\\”, and the files in it must have “.dat” extension. Each file represent a group, the file name is the group name, the content inside are peer names. Each peer name is in one row. After the data is loaded to the map, the tree nodes are created according to the map. Finally the memory of the map is freed. After all real groups showed, an “Unknown Group” will be added. This is how the tree in the “Manage Group“ panel constructed.

The second way is through “FilterTree” function. It filter an existing tree depending on whether the peer represented by the node is online or not. If the peer is online, it keeps the corresponding node, otherwise deletes it. The trees in the “create new session” dialog showed when you are creating a new broadcast session or a monitor session are constructed in this way.

* 1. Session list dialogs

In the “Broadcast Desktop” and the “Monitor Desktop”, the CShareSessionDlg and the CMonitorSessionDlg are behind. I invented two different kinds of sessions, the first is the share session, used in CShareSessionDlg, which means broadcast a peer’s desktop to others in the same session, and the second is monitor session, used in CMonitorSessionDlg, which means monitoring some peers. And the classes “CShareSession” and “CMonitorSession” corresponds to each respectively. And when you are about to create new share session, a dialog containing sharer to choose and viewers to choose from the trees is the UI the CShareSession. Here, in this situation, the CShareSession seems to be the “managed bean” of the dialog as well as the CMonitorSession. The dialogs showed up when creating sessions are modal dialog, before calling the DoModal, I pass an instance of the CShareSession or the CMonitorSession, in this way I can pass in data and let the dialog show, and after the DoModal returned, I can get the instance filled with new data from user input through the dialog. This is an important concept. However in the software I didn’t use this way to pass in data. Note that the concept of monitor session in code level is a little different from that on the UI where a monitor session has only one peer monitored.

1. **Peer** 
   1. Architecture

The architecture is like that of the Control Center. The bottom layer is the CDoorSocket, the mid layer is the CService class, the top layer is the CPeerDlg and the CViewerDlg. Much less complex. Something is different. The dialog corresponding to the CPeerDlg as the main dialog never shows up during the software is running. However, it handles events triggered by tray icon menu or sockets. The CService class provides three kinds of services: socket service, RDP service and system management service. The RDP service mainly includes start RDP service using StartRDPService and stop RDP service using StopRDPService. The system management service here is just install global mouse hook and keyboard hook using SetHook. In the SetHook function I load a library called “GlobalHook” in which global mouse hook and key hook are installed.

* 1. RDP Service

This is the core of the software. Microsoft has provided APIs on <http://msdn.microsoft.com/zh-cn/library/windows/desktop/aa373871(v=vs.85).aspx> , and I also referred to <http://daron.yondem.com/en/post/d8bbab27-169d-43d4-aea7-494845891789> and <http://www.codeproject.com/Articles/29638/Windows-Desktop-Sharing-with-RDP-in-Vista-Add-a-Pr> , the two articles helped me a lot. And I even straightly quoted two classes “CMyRDPSessionEvents” to provide event handle and “B” to transform a normal string to BSTR.

1. **Command system**

|  |  |  |
| --- | --- | --- |
| reply | self-defined message | classes |
| “sharer started” | WM\_MY\_SHARERSTARTED | CShareSessionDlg |
| “sharer not started” | WM\_MY\_SHARERNOTSTARTED | CShareSessionDlg |
| “sharer stopped” | WM\_MY\_SHARERSTOPPED | CShareSessionDlg |
| “sharer not stopped” |  |  |
| “viewer started” |  |  |
| “viewer not started” |  |  |
| “viewer stopped” |  |  |
| “viewer not stopped” |  |  |
| “monitor opened” | WM\_MY\_MONITOROPENED | CMonitorSessionDlg |
| “monitor not opened” | WM\_MY\_MONITORNOTOPENED | CMonitorSessionDlg |
| “monitor closed” | WM\_MY\_MONITORCLOSED | CMonitorSessionDlg |
| “monitor not closed” |  |  |
| client’s command “logname” | WM\_MY\_LOGNAME | CPeerTreeDlg CControlCenterDlg |
| socket close event | WM\_MY\_SOCKCLOSE | CPeerTreeDlg CShareSessionDlg CMonitorSessionDlg CControlCenterDlg |
| client’s command  “hand up” | WM\_MY\_HANDUP | CControlCenterDlg CHandDlg |

Command sent from the server

|  |  |  |  |
| --- | --- | --- | --- |
| functionality | command | success reply | failure reply |
| start RDP service | “start sharer” | sharer started ticket:+ticket string | sharer not started |
| stop RDP service | “stop sharer” | sharer stopped | sharer not stopped |
| let RDPViewer connect RDP sharer according to ticket | “start viewer” + ticket string | viewer started | viewer not started |
| disconnect RDP | “stop viewer” | viewer stopped | viewer not stopped |
| the same as start sharer | “open monitor” | monitor opened ticket:+ticket string | monitor not opened |
| the same as stop sharer | “close monitor” | monitor closed | monitor not closed |
| lock viewer | “lock” |  |  |
| unlock viewer | “unlock” |  |  |
| disable the “Exit program” menu item on client | “config disable exit” |  |  |
| enable the “Exit program” menu item | “config enable exit” |  |  |
| disconnect |  |  |  |

Command sent from the client (no reply from server expected )

|  |  |
| --- | --- |
| functionality | command |
| log on | “logname”+name |
| electronic hand up | “hand up” |
| disconnect |  |

Note: white means nothing.

1. **Language**

In order to internationalize the software, language is a problem. For providing editions of different languages, I transferred every string in the source code to the string table of the resource script. Thus all the language related resources are in the resource script. The default resource script is of Chinese. Collect the resources into another dll project to build a resource-only dll as an independent language package. More information on <http://msdn.microsoft.com/en-us/library/8fkteez0.aspx> .